

引导大语言模型生成计算机可解析内容[1][2]

Guiding Large Language Models to Generate Computer-Parsable Content

王家晔* · 2024/4/22
答辩第四小组 · 指导老师蔡妍

[1]初公开于：https://chinaxiv.org/abs/202403.00340
[2]英文版本：https://arxiv.org/abs/2404.05499
*作者邮箱：hk-shao@outlook.com

大纲 (Outline)

1. 背景 (Background)
2. 动机 (Motivation)
3. 方法 (Method)
4. 效果 (Effect)
5. 展望 (Prospect)
6. 致谢 (ACKs)

背景 (Background)

1. 背景 (Background)
   - 概述 (Overview)
   - 关于本文 (About)
   - 大语言模型 (LLM)
   - 领域特定语言 (DSL)
2. 动机 (Motivation)
3. 方法 (Method)
4. 效果 (Effect)
5. 展望 (Prospect)
6. 致谢 (ACKs)

1.1 概述 (Overview)

关键词：(
 "结构化内容生成",
 "大语言模型",
 "约束解码",
 "元语言",
 "协程",
)

一句话概括全文：**领域特定语言** (DSL) 通用且实用，本研究从**理论**和**实验**两方面说明了**大语言模型** (LLM) 在生成 DSL 的任务上存在**性能缺陷**，因此提出了基于**协程**和 Python **元编程**的元语言 **YieldLang** 作为**约束解码器**来引导 LLM 生成 DSL，实验数据表明本研究取得了**显著的效果**，还有望**降低推理成本**。

1.2 关于本文 (About)

指导：蔡妍 (Yan Cai)
作者：王家晔[1,2] (Jiaye Wang)
单位：[1]华南师范大学 · 软件学院 (School of Software, SCNU), [2]腾讯 · PCG (Platform and Content Group, Tencent Inc.)

Note: 作者在腾讯转正后的**业余时间独自研究**，并于在校期间**独立完成写作**。文章已通过中科院预印本平台[3] (ChinaXiv) 和 arXiv[4] 的审核，诚挚接受大家的批评和指正，欢迎关注与讨论。

[3]初公开于：https://chinaxiv.org/abs/202403.00340
[4]英文版本：https://arxiv.org/abs/2404.05499

1.3 大语言模型 (LLM)

目前流行的 LLM 基于 **Transformer** 架构。

$$P(x) = \prod_{i=1}^{n} p(x_i \mid x_1 \cdot x_2 \cdot \ldots \cdot x_{i-1})$$

$$P(x_{n+1}|x) = p(x_{n+1} \mid x_1 \cdot x_2 \cdot \ldots \cdot x_n)$$

- Transformer: 一种**自回归的生成模型**
  - LLM 的词表：字母表 Σ
  - LLM 所有可能的 Token 串：Σ^*
  - 提示词：串 $x = (x_1 \cdot x_2 \cdot \ldots \cdot x_n) \in \Sigma^*$
  - **下一个 Token 在 Σ 上的概率分布** $P(x_{n+1}|x)$

Note: 本研究暂不讨论 Transformer 的细节。

国外

- **OpenAI** · ChatGPT —— AI 独角兽
  - **2018** GPT-1: **不温不热**
  - **2019** GPT-2: **不温不热**
  - **2020** GPT-3: **不温不热**
  - **2022** 年底 ChatGPT: **火爆全球**
    - **5** 天，注册用户达 **100 万**
    - **2** 个月，活跃用户达一个亿 (**10000 万**)
- **Anthropic** · Claude —— AI 独角兽
- **Google** · Gemini —— 互联网巨头
- **Meta** · LLaMA —— 社交巨头
- **xAI** · Grok —— Elon Musk
- …

国内

- **百度** · 文心一言 —— 互联网大厂
- **阿里** · 通义千问 —— 互联网大厂
- **字节** · 豆包 —— 互联网大厂
- **腾讯** · 混元 —— 互联网大厂
- **商汤** · 商量 —— 科研机构
- **智谱** · 清言 —— 出自高校
- …

1. 各高校、科研机构推出自家大模型
2. 互联网大厂角逐大模型霸主，加速落地
3. **华为**、**小米**、**OPPO**、**vivo** 等各家手机厂商也开始**拥抱大语言模型**为用户提供智能服务

1.4 领域特定语言 (DSL)

类型	例子	用途
编程语言	Python	编写程序
数据格式	JSON	数据交换
配置文件	YAML	配置管理
指令集	x86	芯片指令
协议	HTTP	网络通信

领域特定语言 (Domain Specific Language, DSL)

- 为**特定领域**应用程序设计的计算机语言
  - 能够被计算机准确识别的**结构化字串**
    - 被程序员或程序所**期望的数据结构**

各领域使用 DSL

- **工业界**
  - 编程语言、数据格式、配置文件、指令集、协议
- **学术界**
  - 图文排版、分子结构[1]、蛋白质结构、量子算法[2]
- **商业和艺术领域**
  - COBOL[3]、Csound、ChucK[4] 和 Processing[5] 等

Note: DSL 与通用编程语言 (GPL) 有区别，本研究不强调区别，统一称之为 DSL。

[1]例如使用 ASCII 字符串明确分子结构的 OpenSMILES
[2]例如拥有了良好句法定义的 λ-Q# 是一种领域特定语言
[3]专用于商业的编程语言，也是最早的高级编程语言之一
[4]专用于实时声音合成或音乐创作的计算机编程语言
[5]为电子艺术以及视觉交互设计而创建的编程语言

动机 (Motivation)

1. 背景 (Background)
2. 动机 (Motivation)
   - DSL 异步解析需求
   - DSL 生成性能不好
3. 方法 (Method)
4. 效果 (Effect)
5. 展望 (Prospect)
6. 致谢 (ACKs)

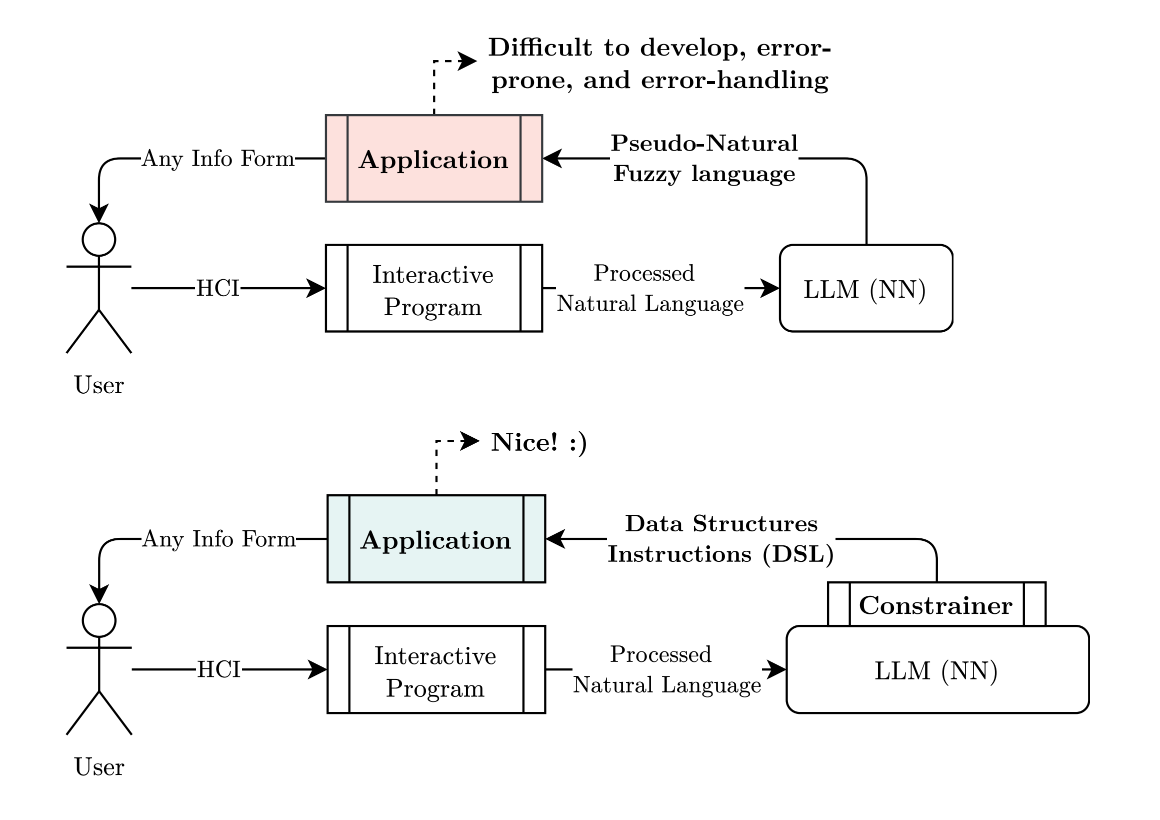
动机 (Motivation)

一句话概括动机：

学术界、工业界近年来有**控制 LLM**[1][2][3][4][5] 和**生成 DSL**[6][7][8][9] 的需求，我的实践表明**现有方法可以改进**。

- 两个需求异曲同工，能够进行统一处理

[1] OpenAI. “More Control”.
[2] https://www.langchain.com
[3] https://arxiv.org/abs/2305.19234
[4] https://arxiv.org/abs/2109.05093
[5] https://github.com/guidance-ai/guidance
[6] https://github.com/mangiucugna/json_repair
[7] https://github.com/vllm-project/vllm/pull/2105
[8] https://github.com/huggingface/transformers/pull/27557
[9] João Lages: OpenAI JSON Mode vs Functions. Medium. 2024



- **上图**：**难题**（**应用程序不易**处理**模糊**的信息）
- **下图**：本研究的**解决方案**（基于约束解码器）

2.3 DSL 异步解析需求

1. **大语言模型补全代码**
   - 输出了 **1000 个 Token** 的 Java 代码，但是因为某个地方**少了一个括号**，导致整个代码无法运行 (**Syntax Error**)，连 **Parser** 都过不了
2. **大文件的传输和解析**
   - 小明传输了一个 **1000 GiB** 的 JSON 数据文件，小红的程序收到后**解析出错**。小红找小明修复，小明发现原来 JSON 末尾**多了一个逗号**

- 为什么要**人工修复**？
- 为什么要**传输完毕之后才能解析**？
- **浪费的算力、带宽和时间**由谁来承担？！

现有方法

1. **大部分时间消耗**在 LLM 的生成（或传输）过程
2. Parser 通常假设给定的串 (DSL) 正确，是**同步**的
    - **鲁棒性不好**，不容易灵活处理错误和恢复解析
    - 增量解析**复杂度下限**是： $O(n^2) = \sum_{i=1}^{n} O(i)$

如果能够**异步**地处理 DSL，那么就可以一边“传输”一边“处理”。若在传输过程中就**发现错误**，就**及时止损**并给予通知，能够**节省带宽和时间**。

```
> JSON.parse(`{ "key": 0`)
✖ ▶Uncaught SyntaxError: Expected ',' or '}' after property VM234:1
value in JSON at position 10 (line 1 column 11)
    at JSON.parse (<anonymous>)
    at <anonymous>:1:6
```

- Microsoft Edge 浏览器无法给予足够精确的提示

2.4 DSL 生成性能不好

设计实验

常见的 DSL 句法特征：**匹配的括号对**。

- 构成语言 $L(G)$, G 是一个 CFG
- 若 $s \in L(G)$ 且 $l =$ '(' 且 $r =$ ')'
- **集合** $I = \{srx \mid \forall s \in L(G), x \in \Sigma^*\}$
  - 全部都是非法的括号对
  - 令 $s = l^n r^n \in L(G)$

(((()))) $x \rightarrow$

(	97.8%
)	2.2%

((((())))) $x \rightarrow$

(	99.9%
)	0.1%

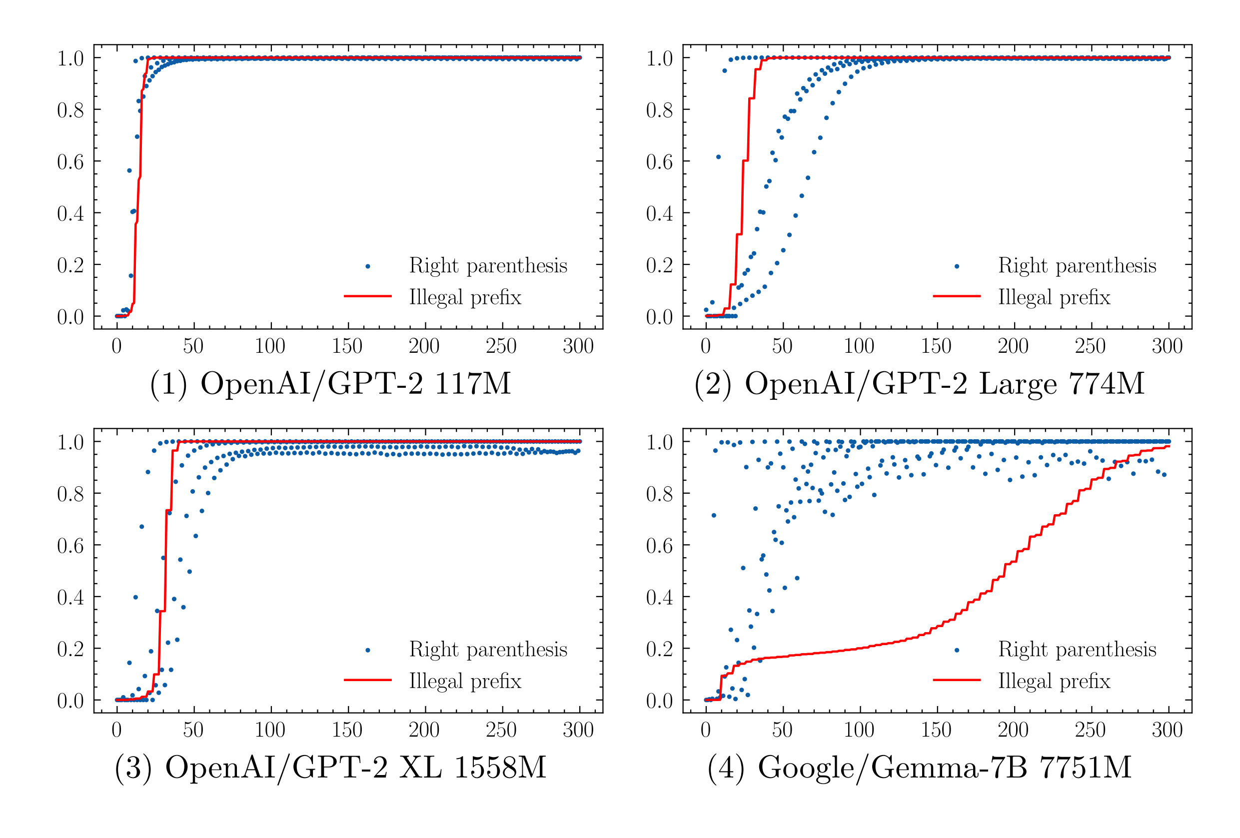
(((((()))))) $x \rightarrow$

(	97.4%
)	0.6%

((((((())))))) $x \rightarrow$

(	97.9%
)	2.1%

性能曲线 (X, Y) : 平均字符串长度 $|s|$, 错误概率 $p(e)$



- **蓝点**：**错误 Token**（右括号）的可能性
- **红线**：**错误 DSL 前缀**（非法括号串）的可能性

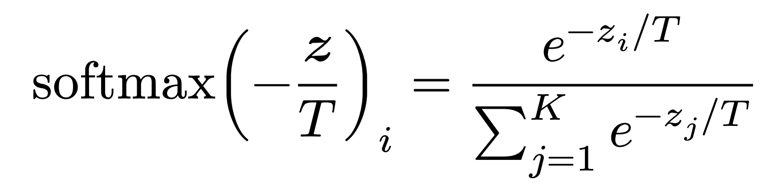
参数量达到近 78 亿的 Gemma 模型，其 $|s|$ **达到 282** 时，**错误率高达 95%** 以上。

- 这些语言模型的 DSL 生成性能是**难以接受**的

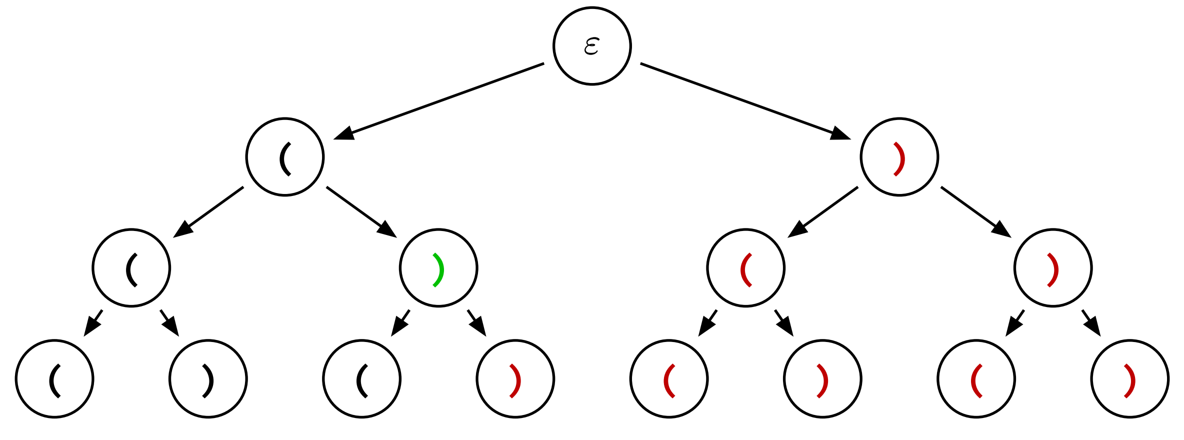
模型名称	**参数量**	Date$^{\text{GMT+8}}$	$\|s\|_{r(e)>50\%}$	$\|s\|_{r(e)>95\%}$
GPT-2	117M	2019-2-14	14	20
GPT-2 Large	774M	2019-2-14	24	32
GPT-2 XL	1558M	2019-2-14	32	36
Gemma	7751M	2024-2-21	194	282

- 四种模型在 $|s|$ 增加时，错误概率 $p(e)$ 不断提升
- **匹配的括号对是非常普遍的 DSL 句法特征**
- 能够生成 DSL 的子集是生成 DSL 的前提
- **LLM 的 DSL 生成性能在快速下降！**

为什么现有 LLM 生成 DSL 的性能不好?



$$\mathrm{softmax}\left(-\frac{z}{T}\right)_i = \frac{e^{-z_i/T}}{\sum_{j=1}^{K} e^{-z_j/T}}$$



- **LLM 的生成是具有随机性的**
  - 基于**自回归模型**获得一个串需要**多次采样**
  - 深度神经网络输出层概率的**可解释性不好**
  - **温度**参数会影响概率分布，**采样有随机性**

方法 (Method)

1. 背景 (Background)
2. 动机 (Motivation)
3. **方法 (Method)**
   - 元语言和协程
   - 约束解码器
   - YieldLang
   - 采样器
4. 效果 (Effect)
5. 展望 (Prospect)
6. 致谢 (ACKs)

3.1 元语言和协程

- **元语言**
  - 用于**表述语言**的语言
  - 本文特指描述 **CFG** 的语言，是一种**形式语言**
    - 被**广泛使用**的、可被**图灵机识别**的语言之一
  - 四元组 $G = (N, \Sigma, P, S)$ 定义了形式文法
    - 有限的非终结符集 N
    - 有限的终结符集（字母表） Σ
    - 有限的产生规则集 P
    - 开始符号 $S \in N$
  - 产生规则
    - 形如 $(\Sigma \cup N)^* N (\Sigma \cup N)^* \to (\Sigma \cup N)^*$

合法的括号对的语言

- 产生规则 P
  - 若 $l =$ '(', $r =$ ')', 开始符号 S, 空串 ε

$$P = \{S \to \varepsilon, S \to lSrS\}$$

- 更清晰的表示

$$\langle \text{Pairs} \rangle \to \langle \text{Pair} \rangle$$
$$\langle \text{Pairs} \rangle \to \langle \text{Pair} \rangle \langle \text{Pairs} \rangle$$
$$\langle \text{Pair} \rangle \to (\ \langle \text{Pairs} \rangle \ )$$
$$\langle \text{Pair} \rangle \to \varepsilon$$

- (), (()), (())(), (()())(), ...

- **C 语言代码的例子**
  - `if ( x > 9 ) { x = 0; y = y + 1; }`

```
  ⟨Stmt⟩ → ⟨Id⟩ = ⟨Expr⟩ ;
  ⟨Stmt⟩ → { ⟨StmtList⟩ }
  ⟨Stmt⟩ → if ( ⟨Expr⟩ ) ⟨Stmt⟩
⟨StmtList⟩ → ⟨Stmt⟩
⟨StmtList⟩ → ⟨StmtList⟩ ⟨Stmt⟩
  ⟨Expr⟩ → ⟨Id⟩
  ⟨Expr⟩ → ⟨Num⟩
  ⟨Expr⟩ → ⟨Expr⟩ ⟨Optr⟩ ⟨Expr⟩
    ⟨Id⟩ → x
    ⟨Id⟩ → y
   ⟨Num⟩ → 0
   ⟨Num⟩ → 1
   ⟨Num⟩ → 9
  ⟨Optr⟩ → >
  ⟨Optr⟩ → +
```

```
⟨Stmt⟩
if ( ⟨Expr⟩ ) ⟨Stmt⟩
if ( ⟨Expr⟩ ⟨Optr⟩ ⟨Expr⟩ ) ⟨Stmt⟩
if ( ⟨Id⟩ ⟨Optr⟩ ⟨Expr⟩ ) ⟨Stmt⟩
if ( x ⟨Optr⟩ ⟨Expr⟩ ) ⟨Stmt⟩
if ( x > ⟨Expr⟩ ) ⟨Stmt⟩
if ( x > ⟨Num⟩ ) ⟨Stmt⟩
if ( x > 9 ) ⟨Stmt⟩
if ( x > 9 ) { ⟨StmtList⟩ }
if ( x > 9 ) { ⟨StmtList⟩ ⟨Stmt⟩ }
if ( x > 9 ) { ⟨Stmt⟩ ⟨Stmt⟩ }
if ( x > 9 ) { ⟨Id⟩ = ⟨Expr⟩ ; ⟨Stmt⟩ }
if ( x > 9 ) { x = ⟨Expr⟩ ; ⟨Stmt⟩ }
if ( x > 9 ) { x = ⟨Num⟩ ; ⟨Stmt⟩ }
if ( x > 9 ) { x = 0 ; ⟨Stmt⟩ }
if ( x > 9 ) { x = 0 ; ⟨Id⟩ = ⟨Expr⟩ ; }
if ( x > 9 ) { x = 0 ; y = ⟨Expr⟩ ; }
if ( x > 9 ) { x = 0 ; y = ⟨Expr⟩ ⟨Optr⟩ ⟨Expr⟩ ; }
if ( x > 9 ) { x = 0 ; y = ⟨Id⟩ ⟨Optr⟩ ⟨Expr⟩ ; }
if ( x > 9 ) { x = 0 ; y = y ⟨Optr⟩ ⟨Expr⟩ ; }
if ( x > 9 ) { x = 0 ; y = y + ⟨Expr⟩ ; }
if ( x > 9 ) { x = 0 ; y = y + ⟨Num⟩ ; }
if ( x > 9 ) { x = 0 ; y = y + 1 ; }
```

- **左侧**：C 语言的一个子集 CFG 表示
- **右侧**：将代码表示为 CFG 的推导树

3.2 约束解码器

$\textit{Constrained-Generation}\ (x \in \Sigma^*, G : \mathrm{CFG})$:

$\hat{y} \leftarrow \varepsilon$ ▷ 初始化空串 ε

while True:

  $\bar{y} \leftarrow \text{decode}\ P_{\mathrm{LLM}}(y \mid x, G, \hat{y}, \ldots)$

  $\hat{y} \leftarrow \hat{y} \cdot \bar{y}$ ▷ 连接串并更新 $\hat{y}$

  if $\hat{y} \in L(G)$**:** ▷ 尝试验证串 $\hat{y} \in L(G)$

    return $\hat{y}$ ▷ 返回预期的 DSL

  else:

    $y_{\text{prefix}}, M_P(y_{\text{prefix}}) \leftarrow \textbf{Generator}(\hat{y}, G)$

    $\omega^* \leftarrow \arg\max P_{\mathrm{LLM}}(\omega \mid \omega \in M_P(y_{\text{prefix}}), y_{\text{prefix}}, \ldots)$

    $\hat{y} \leftarrow y_{\text{prefix}} \cdot \omega^*$ ▷ 更新 $\hat{y} \in P(G)$

- **2021** 首次提出：ElementAI → **EMNLP**
- **2023** 进一步发展：MIT, Google → **NIPS**
- 本研究实现了一种采用**协程**、**异步**的解码器
- **YieldLang** ↔ 元语言框架 ↔ **Generator**

3.3 YieldLang

```
class PairsGenerator(TextGenerator):
    def top(self):
        yield self.pairs
    def pairs(self):
        yield select(
            (self.pair),
            (self.pair, self.pairs)
        )
    def pair(self):
        yield optional(
            '(', self.pairs, ')'
        )
```

- **生成器**：`class TextGenerator`
  - 括号对、浮点数、对象、列表、JSON、流图、…
- **采样器**：`class TextSampler`
  - 随机采样、LLM 采样、自定义处理器、…

- **YieldLang 的 JSON 生成器**
  - 根据 JSON 官方的语法表达[1]

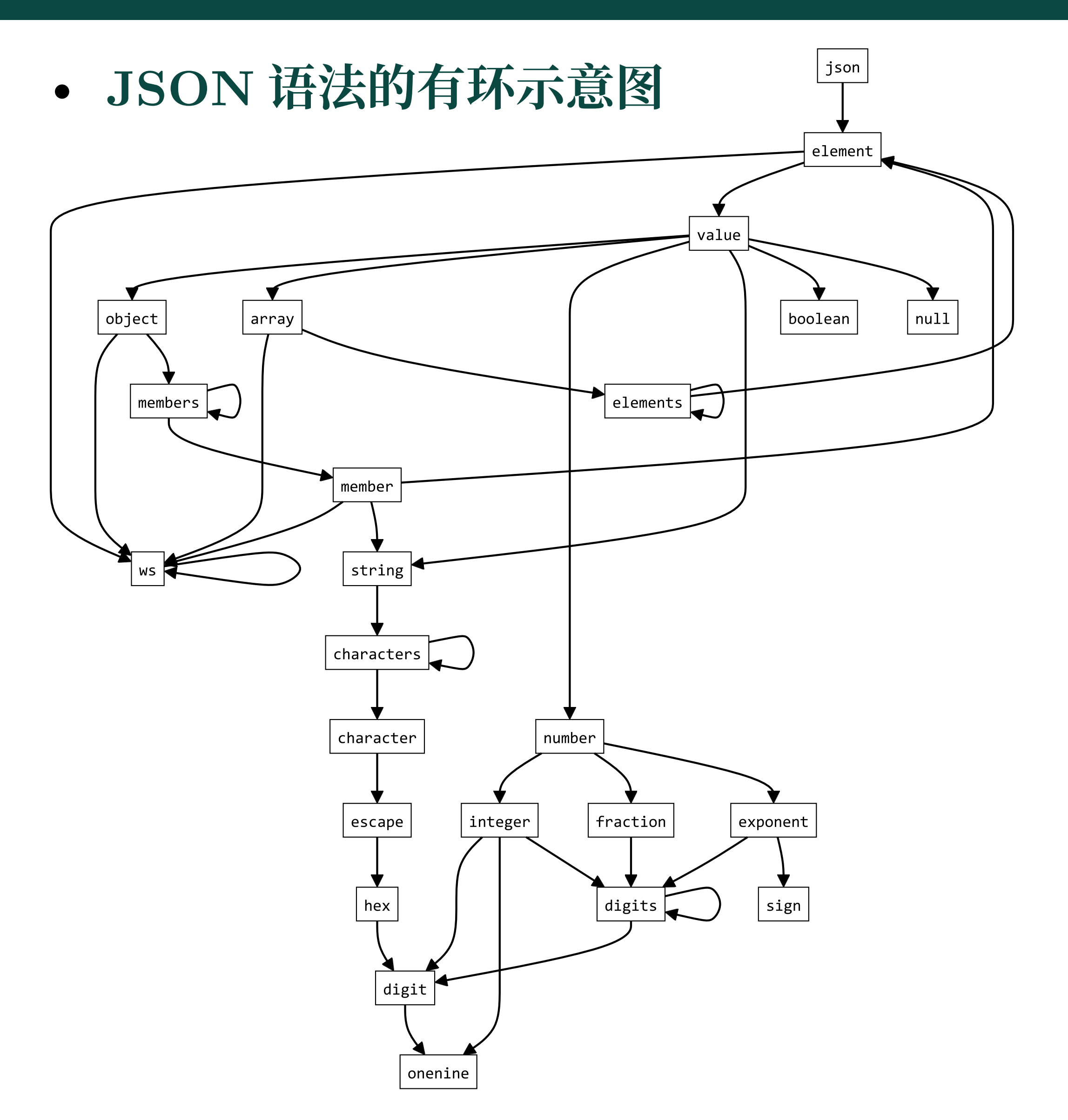
```
class JSONGenerator(TextGenerator):
    def top(self):
        yield self.json
    def json(self):
        yield self.element
    def object(self):
        yield select(
            ('{', self.ws, '}'),
            ('{', self.members, '}')
        )
    def members(self):
        yield select(
            (self.member),
            (self.member, ',', self.members)
        )
    def member(self):
        yield (self.ws, self.string, self.ws, ':', self.element)
    def array(self):
        yield select(
            ('[', self.ws, ']'),
            ('[', self.elements, ']')
        )
    def elements(self):
        yield select(
            (self.element),
```


[1] https://www.json.org/json-en.html

```
        (self.element, ',',  self.elements)
    )
def string(self):
    yield ('"', self.characters, '"')
def characters(self):
    yield optional(self.character, self.characters)
def character(self):
    yield select(
        accept(range=('\u0020', '\uffff'), invalids=('"', '\\')),
        ('\\', self.escape)
    )
def escape(self):
    yield select(
        *'\\"/bfnrt',
        ('u', repeat(self.hex, 4))
    )
def hex(self):
    yield select(
        self.digit,
        select(*'ABCDEF'),
        select(*'abcdef')
    )
def digit(self):
    yield select('0', self.onenine)
def onenine(self):
    yield select(*'123456789')
def number(self):
    yield (self.integer, self.fraction, self.exponent)
def integer(self):
    yield select(
        (self.digit),
        (self.onenine, self.digits),
        ('-', self.digit),
        ('-', self.onenine, self.digits)
    )
def digits(self):
```

```
        yield select(
            (self.digit),
            (self.digit, self.digits)
        )
    def fraction(self):
        yield optional('.', self.digits)
    def exponent(self):
        yield optional(select(
            ('E', self.sign, self.digits),
            ('e', self.sign, self.digits)
        ))
    def sign(self):
        yield optional(select('+', '-'))
    def boolean(self):
        yield select('true', 'false')
    def null(self):
        yield 'null'
    def value(self):
        yield select(
            self.object,
            self.array,
            self.string,
            self.number,
            self.boolean,
            self.null
        )
    def element(self):
        yield (self.ws, self.value, self.ws)
    def ws(self):
        yield optional(select(
            ('\u0020', self.ws),
            ('\u000A', self.ws),
            ('\u000D', self.ws),
            ('\u0009', self.ws)
        ))
```

- **JSON 语法的有环示意图**



- **Yield**
  - 产生
  - 屈服
- **Lang**uage
  - 语言

- **YieldLang**
  - 从形式语言获得灵感、基于协程、支持异步
  - 基本思想：**采样** ⟶ **生成 DSL**
  - **跟踪符号**出入和完成情况，构造 **AST**
    - 即解析 DSL 为 AST (IR)

输入 JSON 上文

JSON 上文

```
1  { "key": 0
```

解析上文

期待的字符/正则

在解析到第 10 个字符时， APNTs 如下：

	期待值 (Char/RegExp)	类型
1	.	Char
2	E	Char
3	e	Char
4	,	Char
5	}	Char
	期待值 (Char/RegExp)	类型

- **YieldLang 的流图生成器（简单例子）**
  - Mermaid[1]是一个图表创建工具和 DSL

```
class MermaidGenerator(TextGenerator):
    def top(self):
        yield self.mermaid
    def mermaid(self):
        match (yield self.graph_name):
            case 'flowchart':
                yield self.flowchart
    def graph_name(self):
        yield select('flowchart')
    def flowchart(self):
        yield (' ', self.flowchart_type, '\n')
        yield join('\n', self.flowchart_rules)
    def flowchart_type(self):
        yield select('TD', 'LR')
    def flowchart_rules(self):
        rand_times = randint(10, 20)
        single_line = ('  '*4, self.flowchart_rule)
        yield from repeat(single_line, rand_times)
    def flowchart_rule(self):
        yield self.node
        yield ' --> '
        yield self.node
    def node(self):
        yield select(*range(1, 10))
```


[1] https://mermaid.js.org/intro/

3.4 采样器

$\mathit{First}\ (A \in N)$:
$s \leftarrow \emptyset$
for $(A \rightarrow \alpha)$ **in** P**:**
  if $\alpha \rightarrow \varepsilon$**:**
    $s \leftarrow s \cup \{\varepsilon\}$
  else if $\alpha = a \in \Sigma$**:**
    $s \leftarrow s \cup \{a\}$
  else if $\alpha = B \in N$**:**
    $s \leftarrow s \cup \mathrm{First}(B)$
return s

- 非终结符 A，$\mathrm{First}(A)$ 由三条规则定义
  1. 若 $A \rightarrow \varepsilon$, 则 $\varepsilon \in \mathrm{First}(A)$
  2. 若 $A \rightarrow a \cdot ... \in \Sigma^*$, 则 $a \in \mathrm{First}(A)$
  3. 若 $A \rightarrow B \cdot ... \in \Sigma^*$, 则 $\mathrm{First}(B) \subseteq \mathrm{First}(A)$

JSON 前缀空字符串 s，什么是 $M_P(s)$?



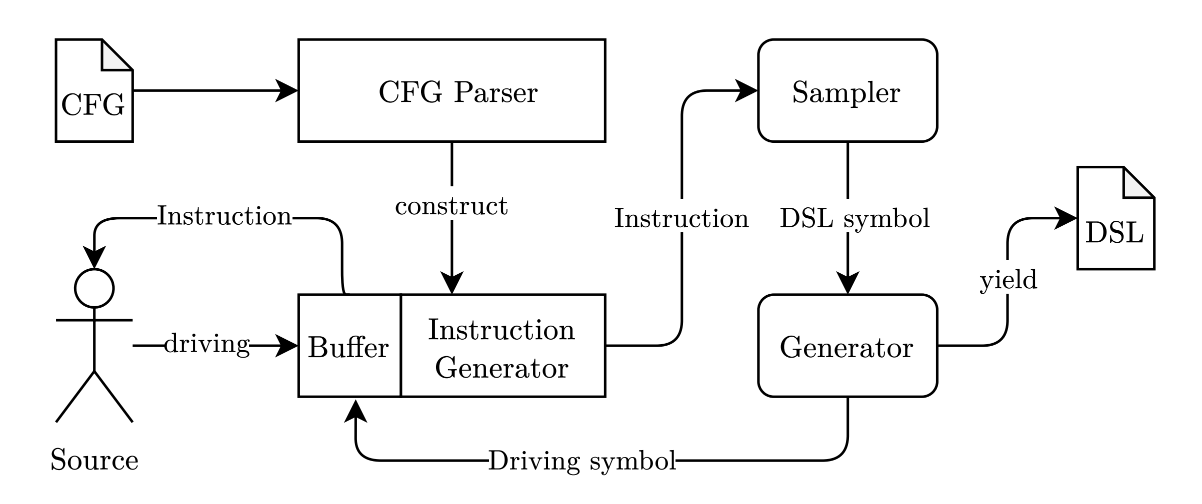
B 在 JSON(JavaScript Object Notation) 上下文中，我们通常会考虑 JSON 的 syntax 规则，以确定哪些字符串或字符可以出现在特定上下文中。JSON 是一种轻量级数据交换格式，基于 JavaScript 对象字面语法，但独立于任何语言，许多编程语言都支持它。 如果我们将空字符串 $s = \varepsilon$ 视为前缀，那么我们要确定所有可能的字符 m ，从而使 sm 成为 JSON 结构的有效前缀。一个 JSON 的结构可以是：

- 对象，以 `'{'` 开头
- 数组，以 `'['` 开头
- 字串，以 `'"'` 开头
- 数字，以 `/0-9/` 或 `'-'` 开头
- 布尔，以 `'t'` (true) 或 `'f'` (false) 开头
- 空值，以 `'n'` 开头

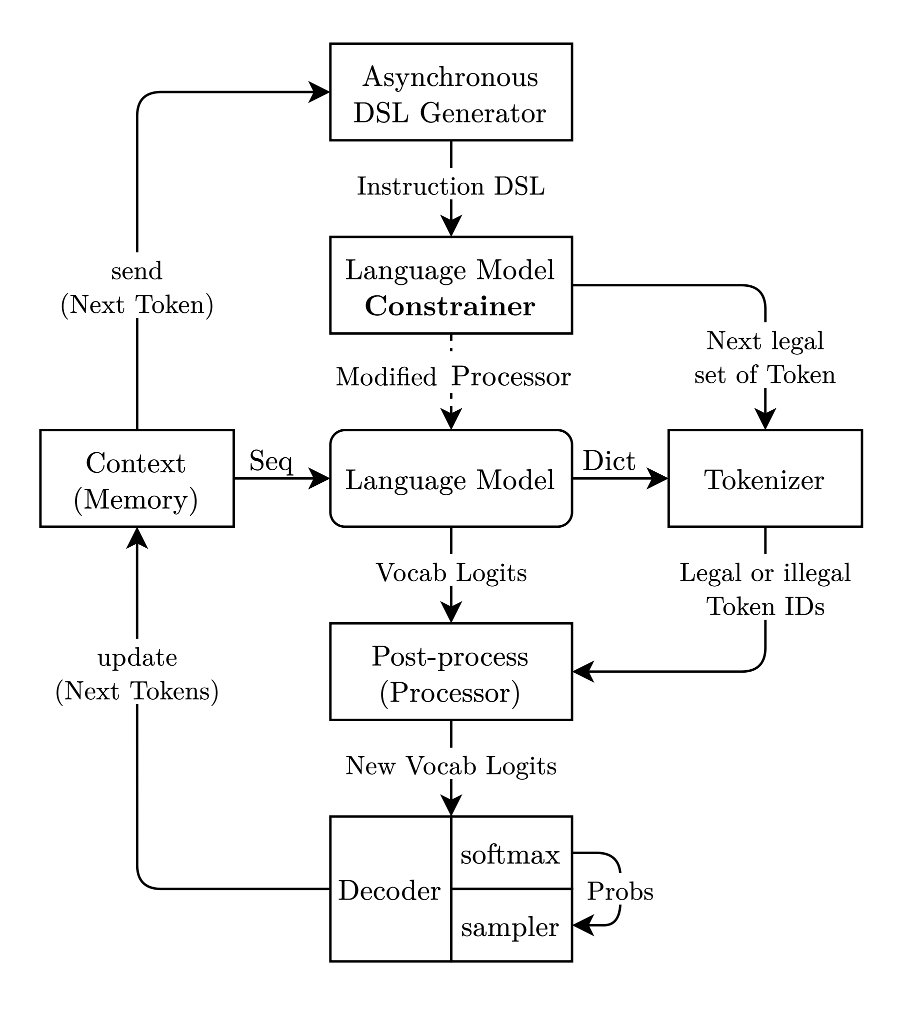
因此，对于空字符串 s ，集合 $M_P(s)$ 将包含所有可以作为 JSON 开头的字符： $M_P(s) =$ {`'{'`, `'['`, `'"'`, `'0'`, `/1-9/`, `'-'`, `'t'`, `'f'`, `'n'`}。这包括所有数字字符和符号 `'-'`，因为负数也是有效的 JSON 数字。请注意，这组字符不包括空格、制表符或其他空白字符，它们虽然可以出现在 JSON 文本中，但并不构成独立的 JSON 结构。

- **前缀**： $P(G) = \{x \in \Sigma^* \mid \exists y \in \Sigma^* : xy \in L(G)\}$
- **采样**： $M_P(s) = \{m \in \Sigma \mid sm \in P(G)\}$

- **一种基于协程的 DSL 生成装置**
  - 关键点在于
    - 异步机制、生成指令
    - **yield**, **yield*** (yield from)



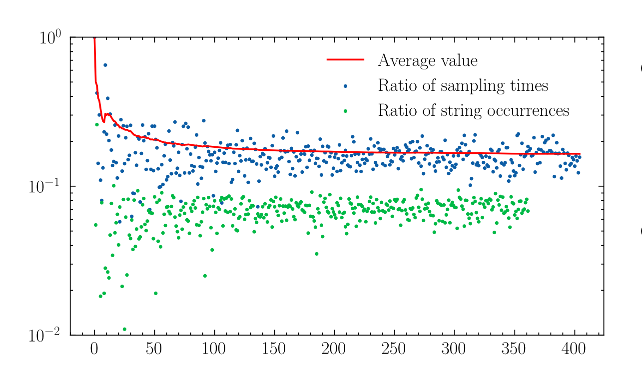
- 两份相关专利已提交官方审核
  - 申请一：**2024102839325**
  - 申请二：**2024102839471**



- 本研究结合**语言模型**的一个**系统流程图**

效果 (Effect)

效果 (Effect)



- 当 $\Sigma_{\text{LLM}} \subseteq \text{Unicode}$
  - **节约 83.5% 采样**
- 可能产生的益处
  - **节约计算资源**

- DSL 生成下游任务上提升 1.09 到 11.6 倍

模型名称	JSON Text		Mermaid		Function Call	
	基准	本文	基准	本文	基准	本文
GPT-2	6.7%	**12.1%**	7.2%	**83.6%**	16.7%	**18.9%**
GPT-2 XL	13.5%	**19.6%**	11.3%	**87.4%**	19.0%	**20.7%**
Gemma-2B	20.4%	**42.1%**	23.2%	**91.1%**	23.7%	**26.4%**
Gemma-7B	29.3%	**49.9%**	34.4%	**97.7%**	28.2%	**31.9%**

展望 (Prospect)

1. 背景 (Background)
2. 动机 (Motivation)
3. 方法 (Method)
4. 效果 (Effect)
5. **展望 (Prospect)**
6. 致谢 (ACKs)

展望 (Prospect)

- 相关方法“教会”AI 使用工具
  - **可能是 AI → AGI 的途径之一**
- LLM 领域很“卷”
  - GPL 用 **DSL 采样**“适配”LLM[1]
  - 相关研究还在不断出现...
- 本研究不足之处或展望
  - YieldLang 的**易用性**可以进一步论证
  - 约束解码在**更多任务**上的条件概率
  - 是否有益于 LLM 的**预训练**或**微调**
  - 本研究装置的**性能表现**需要分析

[1] https://www.moonbitlang.com/blog/moonbit-ai

致谢 (ACKs)

1. 背景 (Background)
2. 动机 (Motivation)
3. 方法 (Method)
4. 效果 (Effect)
5. 展望 (Prospect)
6. **致谢 (ACKs)**

致谢 (ACKs)

1. **感谢上海交通大学和英特尔的 Hansimov**[1]
2. 感谢软件协会 (SoCoding[2]) 、香农和椰社[3]
3. 感谢 Joseph Pan[4] 支持我的相关工作
4. 感谢 Typst[5]、NewCM[6]、Noto[7]
    - Paper 和 Slides 采用 Typst 语言排版
    - Slides 的字体采用 NewCM 和 Noto
5. **感谢前人、亲人、学校和指导老师**

[1]https://github.com/Hansimov
[2]https://socoding.cn/
[3]https://socoding.cn/organization
[4]https://github.com/wzpan
[5]https://github.com/typst/typst
[6]https://git.gnu.org.ua/newcm.git
[7]https://fonts.google.com/noto

参考文献[1,52]

[1] LUNDBERG S, RIBEIRO M T C, EDGAR R, et al. guidance-ai/guidance[M/OL]. 2024. https://github.com/guidance-ai/guidance.

[2] RADFORD A, NARASIMHAN K, SALIMANS T, et al. Improving language understanding by generative pre-training[J]. 2018.

[3] YAO S, ZHAO J, YU D, et al. ReAct: Synergizing Reasoning and Acting in Language Models[Z]. 2023.

[4] SCHOLAK T, SCHUCHER N, BAHDANAU D. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models[C/OL]//Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 2021: 9895-9901. https://aclanthology.org/2021.emnlp-main.779.

[5] OPENAI. Function Calling: Learn how to connect large language models to external tools[Z]. 2024.

[6] LAGES J. OpenAI JSON Mode vs Functions[Z]. 2024.

[7] WANG B, WANG Z, WANG X, et al. Grammar prompting for domain-specific language generation with large language models[J]. Advances in Neural Information Processing Systems, 2024, 36.

[8] WOLF T, GUGGER S, DEBUT L, et al. huggingface/transformers[M/OL]. 2024. https://github.com/huggingface/transformers.

[9] BACCIANELLA S, POWLEY B, TERRY. mangiucugna/json_repair[M/OL]. 2024. https://github.com/mangiucugna/json_repair.

[10] KWON W, LI Z, BAUM A, et al. vllm-project/vllm[M/OL]. 2024. https://github.com/vllm-project/vllm.

[11] GENG S, JOSIFOSKI M, PEYRARD M, et al. Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning[Z]. 2024.

[12] SHINAN E, MEGAING, CHANICPANIC, et al. lark-parser/lark[M/OL]. 2024. https://github.com/lark-parser/lark.

[13] LIN Y, LIN H, XIONG W, et al. Mitigating the Alignment Tax of RLHF[Z]. 2024.

[14] SAMMET J E. Programming languages: History and fundamentals[M]. Prentice-Hall, Inc., 1969.

[15] WEXELBLAT R L. History of programming languages[M]. Academic Press, 1981.

[16] BRUNSFELD M, HLYNSKYI A, QURESHI A, et al. tree-sitter/tree-sitter: v0.22.2[M/OL]. Zenodo, 2024. https://zenodo.org/doi/10.5281/zenodo.10827268. DOI:10.5281/ZENODO.10827268.

[17] BRAY T. The JavaScript Object Notation (JSON) Data Interchange Format[EB/OL]. RFC Editor, 2014. https://www.rfc-editor.org/info/rfc7159. DOI:10.17487/RFC7159.

[18] MASINTER L M, CONNOLLY D W. The 'text/html' Media Type[EB/OL]. RFC Editor, 2000. https://www.rfc-editor.org/info/rfc2854. DOI:10.17487/RFC2854.

[19] ST.LAURENT S, MAKOTO M, KOHN D. XML Media Types[EB/OL]. RFC Editor, 2001. https://www.rfc-editor.org/info/rfc3023. DOI:10.17487/RFC3023.

[20] SHAFRANOVICH Y. Common Format and MIME Type for Comma-Separated Values (CSV) Files[EB/OL]. RFC Editor, 2005. https://www.rfc-editor.org/info/rfc4180. DOI:10.17487/RFC4180.

[21] SVEIDQVIST K, MERMAID C to. Mermaid: Generate diagrams from markdown-like text[EB/OL]. (2014-12-02). https://mermaid.js.org/.

[22] WEININGER D. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules[J]. Journal of chemical information and computer sciences, 1988, 28(1): 31-36.

[23] ION P, MINER R, AUSBROOKS R, et al. Mathematical Markup Language (MathML) Version 3.0[J]. World Wide Web Consortium, 1998.

[24] O'BOYLE N, DALKE A. DeepSMILES: an adaptation of SMILES for use in machine-learning of chemical structures[J]. 2018.

[25] FREED N, BORENSTEIN D N S. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies[EB/OL]. RFC Editor, 1996. https://www.rfc-editor.org/info/rfc2045. DOI:10.17487/RFC2045.

[26] VANDERMEERSCH T, MAYFIELD J, HUTCHISON G, et al. opensmiles/OpenSMILES[M/OL]. 2021. https://github.com/opensmiles/OpenSMILES.

[27] SINGHAL K, HIETALA K, MARSHALL S, et al. Q# as a Quantum Algorithmic Language[J/OL]. Electronic Proceedings in Theoretical Computer

Science, 2023, 394: 170-191. http://dx.doi.org/10.4204/EPTCS.394.10. DOI:10.4204/eptcs.394.10.

[28] OFER D, BRANDES N, LINIAL M. The language of proteins: NLP, machine learning & protein sequences[J/OL]. Computational and Structural Biotechnology Journal, 2021, 19: 1750-1758. https://www.sciencedirect.com/science/article/pii/S2001037021000945. DOI:https://doi.org/10.1016/j.csbj.2021.03.022.

[29] WANG G, COOK P R, SALAZAR S. ChucK: A Strongly Timed Computer Music Language[J/OL]. Computer Music Journal, 2015, 39(4): 10-29. https://doi.org/10.1162/COMJ/_a/_00324. DOI:10.1162/COMJ_a_00324.

[30] LAZZARINI V, YI S, HEINTZ J, et al. Csound: a sound and music computing system[M]. Springer, 2016.

[31] BOHNACKER H, GROSS B, LAUB J, et al. Generative design: visualize, program, and create with processing[M]. Princeton Architectural Press, 2012.

[32] MÄDJE L. A Programmable Markup Language for Typesetting[D]. 2022.

[33] TEAM G, MESNARD T, HARDIN C, et al. Gemma: Open Models Based on Gemini Research and Technology[J]. arXiv preprint arXiv:2403.08295, 2024.

[34] RADFORD A, WU J, CHILD R, et al. Language models are unsupervised multitask learners[J]. OpenAI blog, 2019, 1(8): 9-10.

[35] WOLFRAM S. What is ChatGPT doing ... and why does it work?[EB/OL]. Stephen Wolfram Writings, 2023[2023-02-14]. https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work.

[36] CHOMSKY N. Three models for the description of language[J]. IRE Transactions on information theory, 1956, 2(3): 113-124.

[37] MASCARENHAS F, MEDEIROS S, IERUSALIMSCHY R. On the relation between context-free grammars and parsing expression grammars[J/OL]. Science of Computer Programming, 2014, 89: 235-250. https://www.sciencedirect.com/science/article/pii/S0167642314000276. DOI:https://doi.org/10.1016/j.scico.2014.01.012.

[38] BURGHARDT J. Shows a simplified excerpt of the formal grammar for the C programming language, and a derivation of a piece of C code[EB/OL]. (2020). https://commons.wikimedia.org/wiki/File:C_grammar_stmt.pdf.

[39] KERNIGHAN B W, RITCHIE D M. The C Programming Language[M]. 2nd ed. Englewood Cliffs/NJ: Prentice Hall, 1988.

[40] CROCKER D, OVERELL P. Augmented BNF for Syntax Specifications: ABNF[EB/OL]. RFC Editor, 2008. https://www.rfc-editor.org/info/rfc5234. DOI:10.17487/RFC5234.

[41] NEWYSTATS. Plot of the probability density function of the exponential distribution[Z]. 2019.

[42] ACHIAM J, ADLER S, AGARWAL S, et al. GPT-4 technical report[J]. arXiv preprint arXiv:2303.08774, 2023.

[43] TOUVRON H, MARTIN L, STONE K, et al. Llama 2: Open foundation and fine-tuned chat models[J]. arXiv preprint arXiv:2307.09288, 2023.

[44] WANG C, LIU X, AWADALLAH A H. Cost-effective hyperparameter optimization for large language model generation inference[C]//International Conference on Automated Machine Learning. 2023: 21-21.

[45] RENZE M, GUVEN E. The Effect of Sampling Temperature on Problem Solving in Large Language Models[J]. arXiv preprint arXiv:2402.05201, 2024.

[46] NIELSEN H, MOGUL J, MASINTER L M, et al. Hypertext Transfer Protocol – HTTP/1.1[EB/OL]. RFC Editor, 1999. https://www.rfc-editor.org/info/rfc2616. DOI:10.17487/RFC2616.

[47] FROST R A, HAFIZ R. A new top-down parsing algorithm to accommodate ambiguity and left recursion in polynomial time[J/OL]. SIGPLAN Not., 2006, 41(5): 46-54. https://doi.org/10.1145/1149982.1149988. DOI:10.1145/1149982.1149988.

[48] DAVIES A. Async in C# 5.0[M]. " O'Reilly Media, Inc.", 2012.

[49] DAHL O. CAR Hoare Hierarchical Program Structures In: Structured Programming (Dahl, Dijkstra, Hoare)[M]. Academic Press, 1972.

[50] FOUNDATION P S. The Python Language Reference - Yield expressions[EB/OL]. (2019). https://docs.python.org/3/reference/expressions.html#yieldexpr.

[51] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.

[52] FEI H, ZHANG Y, ZHANG H, et al. MoonBit: Explore the Design of an AI-Friendly Programming Language[J/OL]. 2024. https://www.moonbitlang.com/blog/moonbit-ai.